CENG 3420
Computer Organization & Design

Lecture 16: Cache-2

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

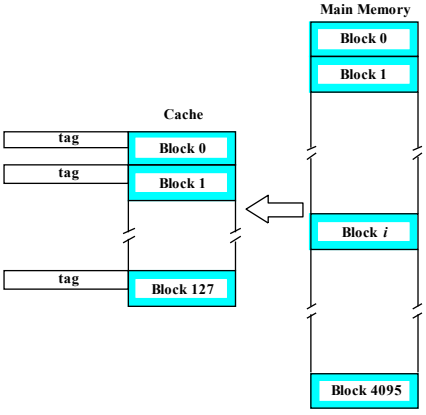
(Textbook: Chapters 5.3–5.4)

Spring 2022

Overview

Associative Mapping

Associative Mapping

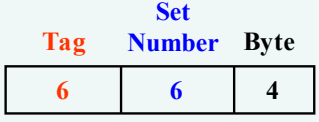


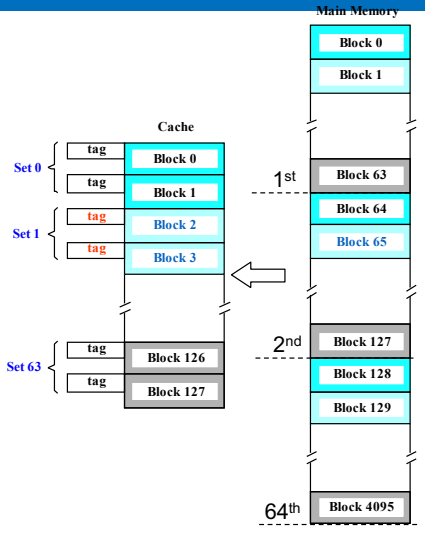
- An MM block can be in **arbitrary** Cache block location
- In this example, all 128 tag entries must be compared with the address Tag in parallel (by hardware)

Associative Mapping Example

1. CPU is looking for [A7B4] MAR = 101001111011**0100**
2. See if the tag 101001111011 matches one of the 128 cache tags
3. If YES, cache hit!
4. Otherwise, get the block into BINGO cache row

Set Associative Mapping





Example: 2-way set associative

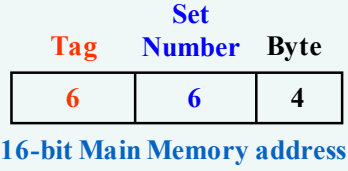
- Combination of direct and associative
- (j mod 64) derives the Set Number
- A cache with k-blocks per set is called a k-way set associative cache.

Notations

Set : how to partition the cache (e.g. 2-set means cache is divided into 2 parts)

Way : # of block in one set (e.g. 2-way means one set has 2 blocks)

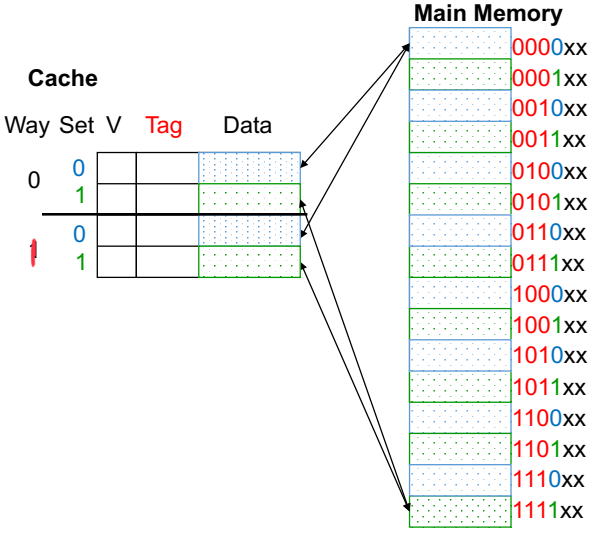
Set Associative Mapping Example 1



E.g. 2-Way Set Associative:

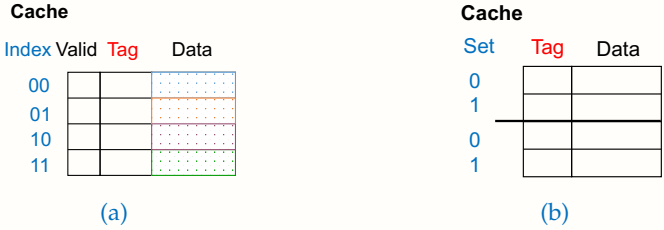
1. CPU is looking for [A7B4] MAR = 1010011110110100
2. Go to cache Set 111011 (59_{10})
   - Block 1110110 (118_{10})
   - Block 1110111 (119_{10})
3. See if ONE of the TWO tags in the Set 111011 is 101001
4. If YES, cache hit!
5. Get the block into BINGO cache row

Set Associative Mapping Example 2



Question: Direct Mapping v.s. 2-Way Set Associate

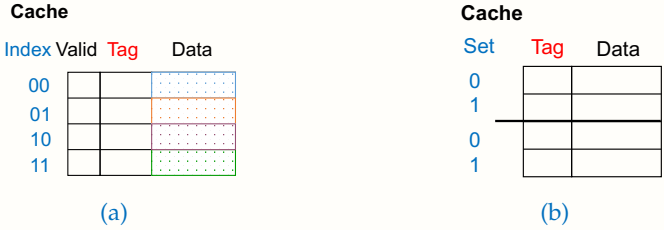
Consider the following two empty caches, calculate Cache hit rates for the reference word addresses: "0 4 0 4 0 4 0 4"



(a) Direct Mapping; (b) 2-Way Set Associative.

Question: Direct Mapping v.s. 2-Way Set Associate

Consider the following two empty caches, calculate Cache hit rates for the reference word addresses: "0 4 0 4 0 4 0 4"

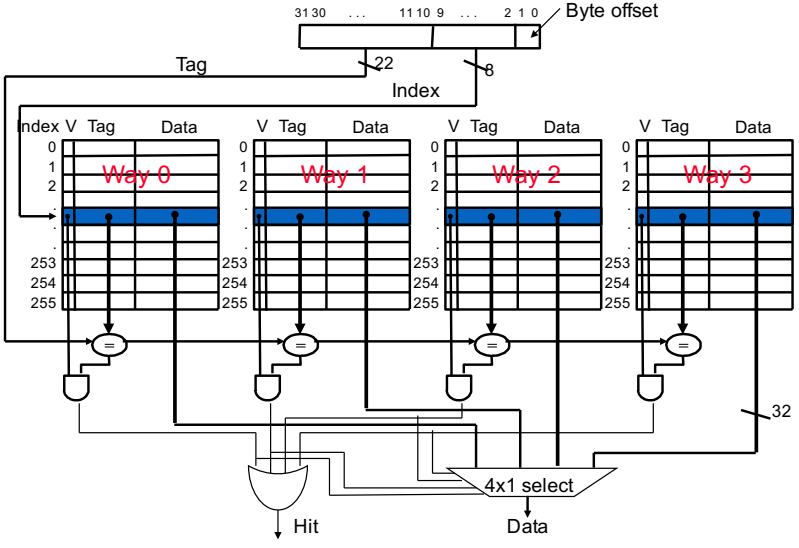


(a) Direct Mapping; (b) 2-Way Set Associative.

- Direct mapping: 0 hit (Ping pong effect)
- 2-Way Set Associative: 6 hits

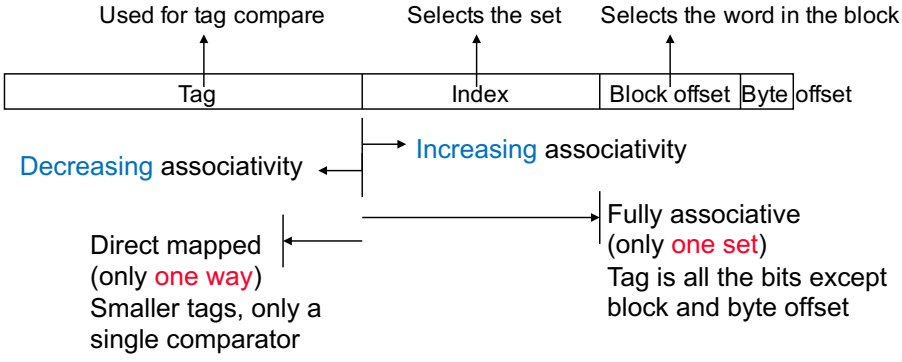
Set Associative Mapping Example 3: MIPS

- $2^8 = 256$ sets each with four ways (each with one block).
- four tags in the set are compared in parallel.



Range of Set Associative Caches

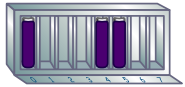
For a fixed size cache:



- Enjoy and refresh the enjoyable memories
- Let go of painful memories

Just as bookshelves come in different shapes and sizes, caches can also take on a variety of forms and capacities. But no matter how large or small they are, caches fall into one of three categories: direct mapped, n-way set associative, and fully associative.

Direct Mapped



A cache block can only go in one spot in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks.

2-Way Set Associative

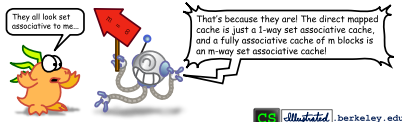
This cache is made up of sets that can fit two blocks each. The index is now used to find the set, and the tag helps find the block within the set.

4-Way Set Associative

Each set here fits four blocks, so there are fewer sets. As such, fewer index bits are needed.

Fully Associative

No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!



Replacement

Handling Cache Read

- `I$` and `D$`
- Read hit: what we want!
- Read miss: stall the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume.

Handling Cache Write Hits

Only `D$`

Case 1: Write-Through

- Cache and memory to be consistent
- always write the data into both the cache block and the next level in the memory hierarchy
- Speed-up: use write buffer and stall only when buffer is full

Case 2: Write-Back

- Write the data only into the cache block
- Write to memory hierarchy when that cache block is "evicted"
- Need a dirty bit for each data cache block

Handling Cache Write Misses

Case 1: Write-Through caches with a write buffer

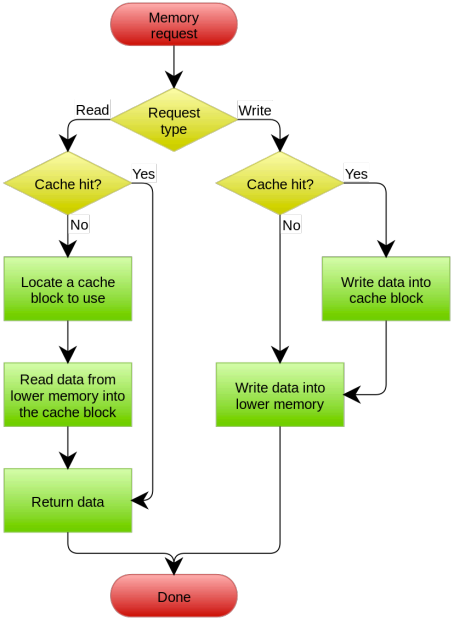
- No-write allocate[1]
- skip cache write (but must invalidate that cache block since it now holds stale data)
- just write the word to the write buffer (and eventually to the next memory level)
- no need to stall if the write buffer isn't full

Case 2: Write-Back caches

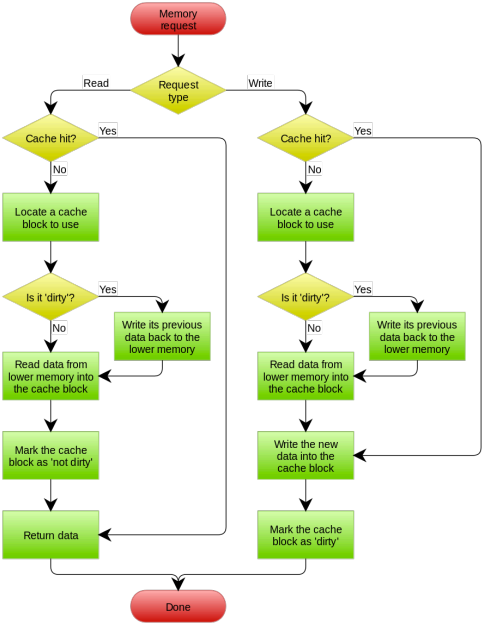
- Write allocate[2]
- Just write the word into the cache updating both the tag and data
- no need to check for cache hit
- no need to stall

[1] The block is modified in the main memory and not loaded into the cache.
[2] The block is loaded on a write miss, followed by the write-hit action.

Write-Through Cache with No-Write Allocation



Write-Back Cache with Write Allocation



Replacement Algorithms

Direct Mapping

- Position of each block fixed
- Whenever replacement is needed (i.e. cache miss $\rightarrow$ new block to load), the choice is obvious and thus no "replacement algorithm" is needed

Associative and Set Associative

- Need to decide which block to replace
- Keep/retain ones likely to be used in near future again

Associative & Set Associative Replacement

Strategy 1: Least Recently Used (LRU)

- e.g. for a 4-block/set cache, use a $\log_2 4 = 2$ bit counter for each block
- Reset the counter to 0 whenever the block is accessed
- counters of other blocks in the same set should be incremented
- On cache miss, replace/ uncache a block with counter reaching 3

Associative & Set Associative Replacement

Strategy 1: Least Recently Used (LRU)

- e.g. for a 4-block/set cache, use a $\log_2 4 = 2$ bit counter for each block
- Reset the counter to 0 whenever the block is accessed
- counters of other blocks in the same set should be incremented
- On cache miss, replace/ uncache a block with counter reaching 3

Strategy 2: Random Replacement

- Choose random block
- ☺Easier to implement at high speed

Conclusion

- Cache Organizations:
  Direct, Associative, Set-Associative
- Cache Replacement Algorithms:
  Random, Least Recently Used
- Cache Hit and Miss Penalty